Uploading an Assignment via VPL System Practitioner Side Manual

Introduction

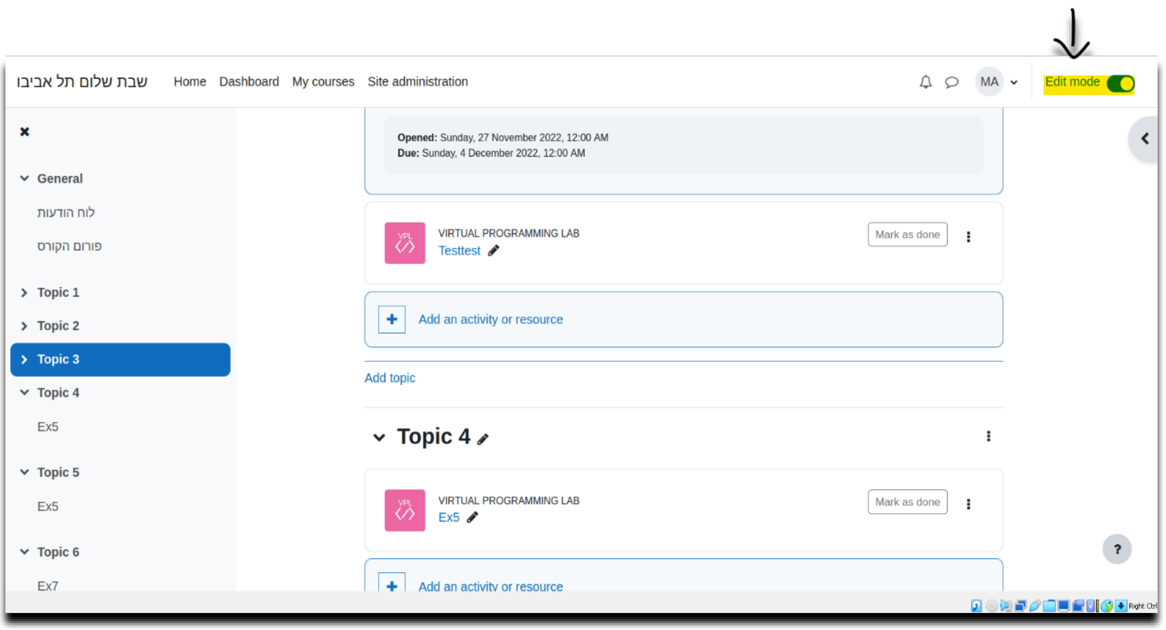
This manual presents a detailed guide for practitioners to upload programming assignments using the Virtual Programming Lab (VPL) system. Whether utilizing a GitHub URL or a Zip file, this process caters to programming languages such as C, C++, Python, or C#.

Step 1: Log in to Moodle

1. Access the Moodle website.
2. Navigate to the specific course where you intend to upload the assignment.

Step 2: Enable Edit Mode

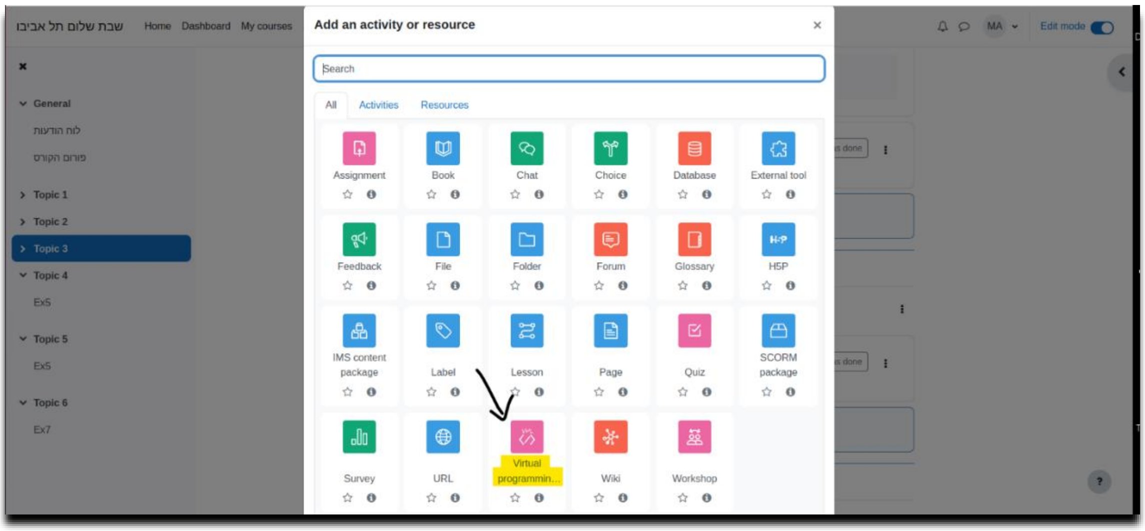
Prior to making any changes, ensure you're in edit mode to conveniently modify the course content.



Opened: Sunday, 27 November 2022, 12:00 AM
Due: Sunday, 4 December 2022, 12:00 AM

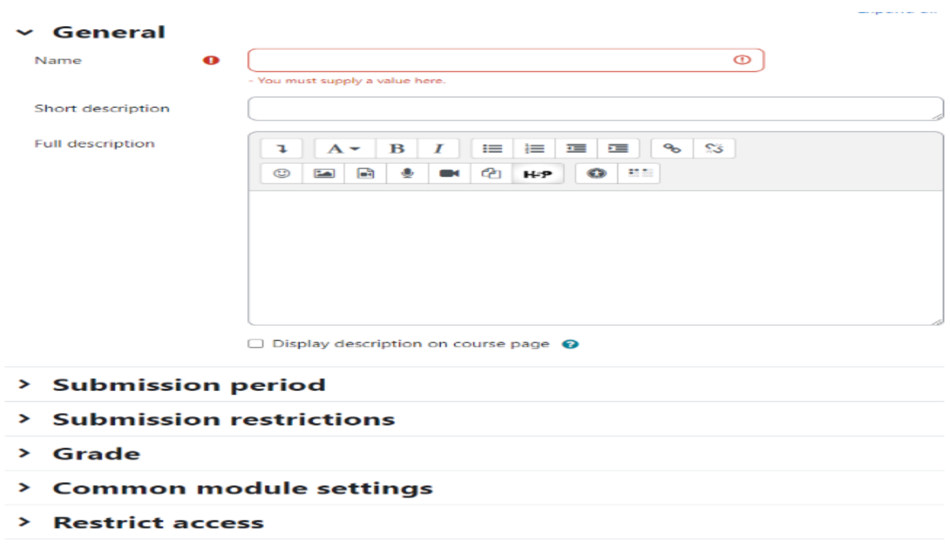
Step 3: Select Virtual Programming Language Option

1. Opt for adding a new activity or resource within the course.
2. Choose the "Virtual Programming Language" option for the new assignment upload.



Step 4: Provide Assignment Details

1. Input all essential assignment information on the subsequent page. Include the assignment name, due date, and pertinent details.
2. Specify the number of files required for the assignment.
3. For group assignments, choose "Group work" under submission restrictions. (Refer to Step 5 for group assignment setup)
4. Click the "Save and Display" button to proceed.



General

Name
- You must supply a value here.

Short description

Full description

Display description on course page

Submission period

Submission restrictions

Grade

Common module settings

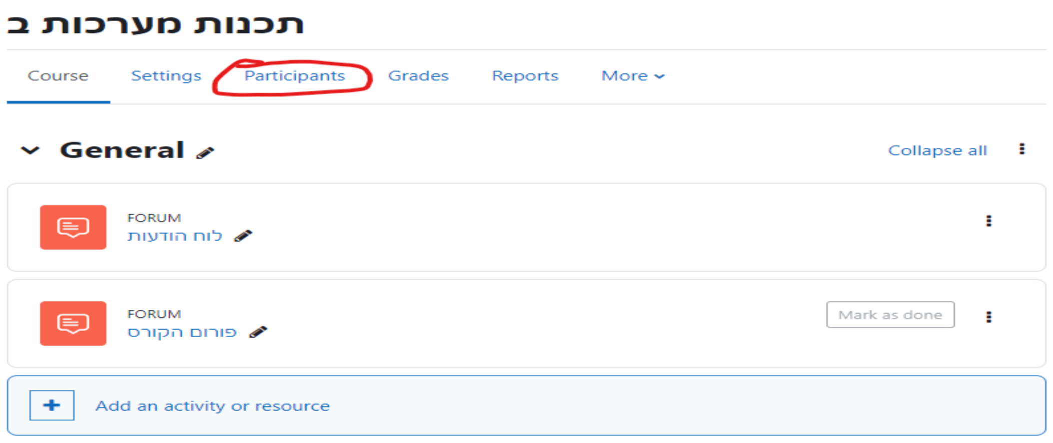
Restrict access

5. **Step 5: Setting Up Groups (Applicable to Group Assignments)**

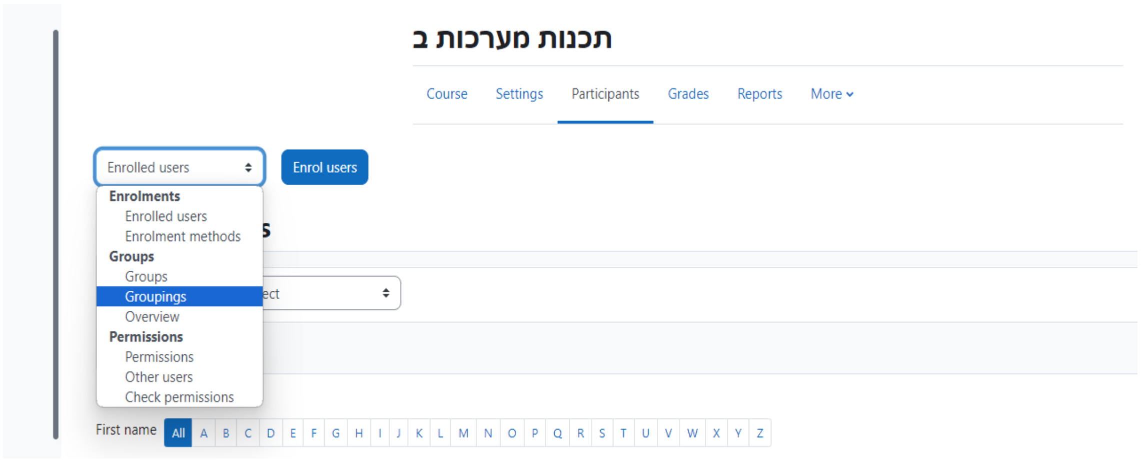
Note: Skip this step for individual assignments.

If your task involves group assignments, follow these steps to effectively configure groups:

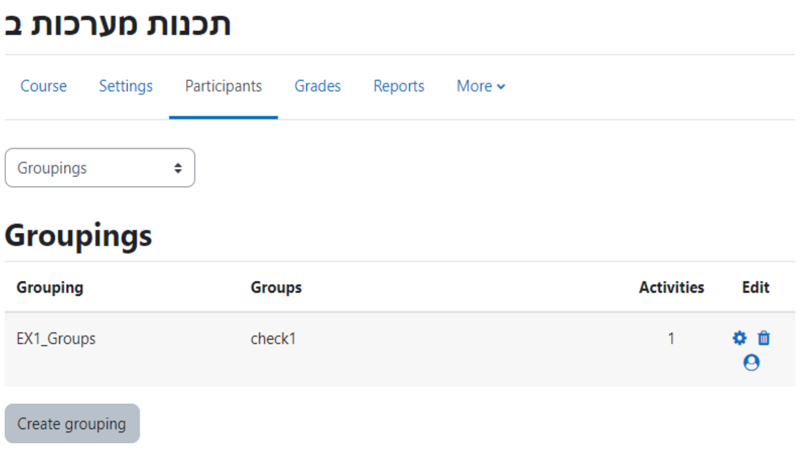
1. Access the "Participants" tab on the course homepage.



2. Select the "Grouping" option from the slide-out list of enrolled users.



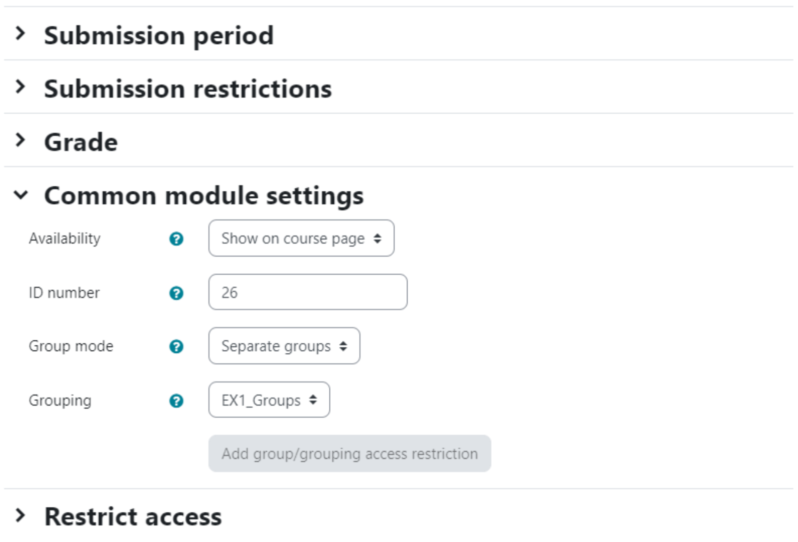
3. Create a new grouping by clicking the "Create grouping" button and assigning a name like "EX1_Groups" for the initial task.


תכנות מערכות ב

Groupings

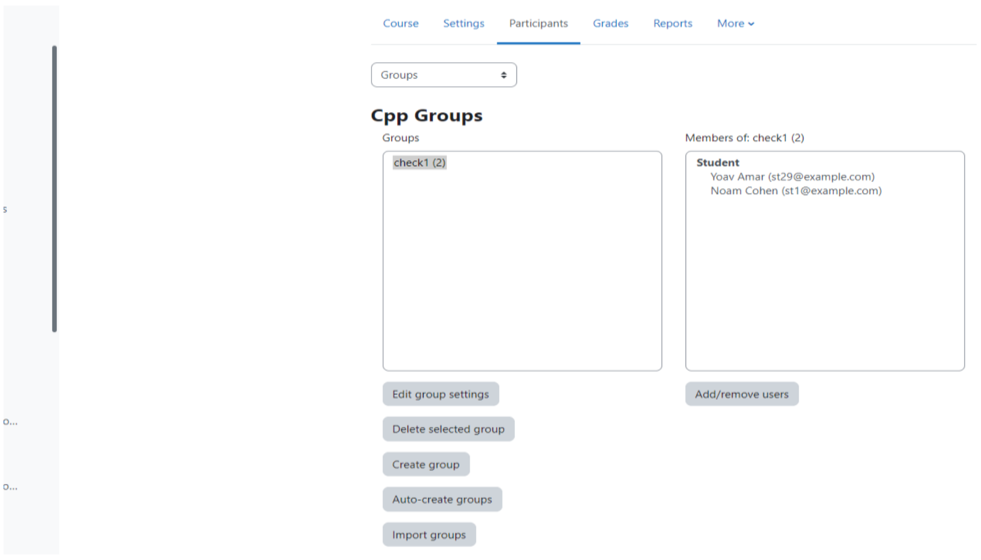
Grouping	Groups	Activities	Edit
EX1_Groups	check1	1	

4. Associate the grouping with the assignment by selecting "Group mode" by choice in the "Common module settings" section. Choose the grouping you created, such as "EX1_Groups."



Availability	Show on course page
ID number	26
Group mode	Separate groups
Grouping	EX1_Groups

5. Assign Users to Groups: To efficiently organize users into groups, you have several options:
    - Import Groups from CSV File: Use a CSV file (e.g., groups_example.csv) to import group names and user lists.
    - Utilize Auto-Create Groups: Automatically generate groups based on criteria like group size or total participants. Assign users randomly, by ID, by name, or create empty groups.
    - Manual Group Creation: Manually create each group, assigning a suitable name.
    - Customize Group Membership: Adjust groups by adding/removing users as needed.



Cpp Groups

Groups: check1 (2)

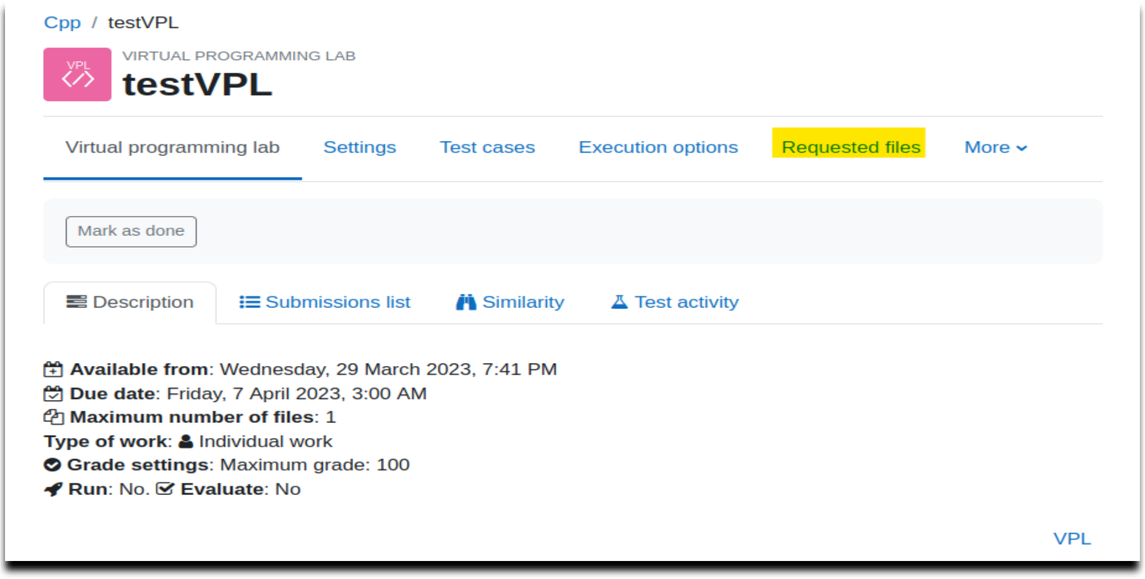
Members of: check1 (2)

Student
Yoav Amar (st29@example.com)
Noam Cohen (st1@example.com)

By leveraging these options, you can effectively structure your groups, aligning them with your educational goals and the collaborative nature of your assignments.

Step 6: Configure Submission Type

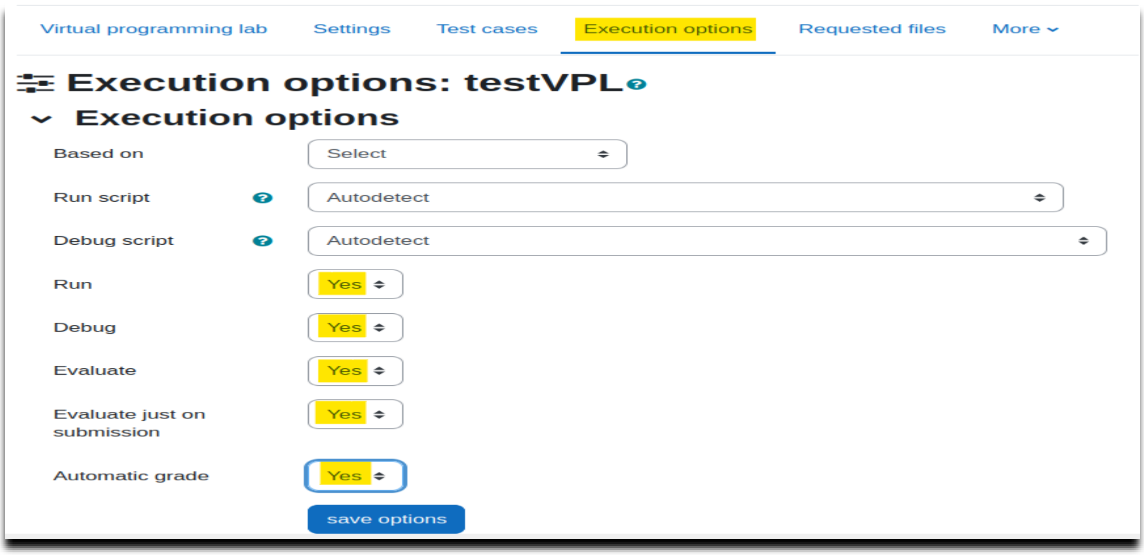
1. Within the assignment settings, click on the "Requested file" tab.
2. Define the submission requirements:
   - For Git submissions: Create a text file (e.g., file.txt) and instruct students to provide a repository link.
   - For Zip file submissions: Prepare a zip file (e.g., files.zip) for students to upload a compressed directory containing necessary files.
3. Save changes (Ctrl + S or applicable option).



Available from: Wednesday, 29 March 2023, 7:41 PM
Due date: Friday, 7 April 2023, 3:00 AM
Maximum number of files: 1
Type of work: Individual work
Grade settings: Maximum grade: 100
Run: No. **Evaluate**: No

Step 7: Configure Execution Options

1. In the assignment settings, go to the "Execution Options" section.
2. Modify request modes to suit your preferences:
   - Run: Enable running programs during submission.
   - Debug: Allow debugging during submission.
   - Evaluate: Permit program evaluation on assignment upload.
   - Evaluate Just on Submission: Evaluate programs only on submission.
   - Automatic Grade: Enable automatic grading by the VPL jail server.


Virtual programming lab | Settings | Test cases | Execution options | Requested files | More

Execution options: testVPL

Execution options

Based on: Select

Run script: Autodetect

Debug script: Autodetect

Run: Yes

Debug: Yes

Evaluate: Yes

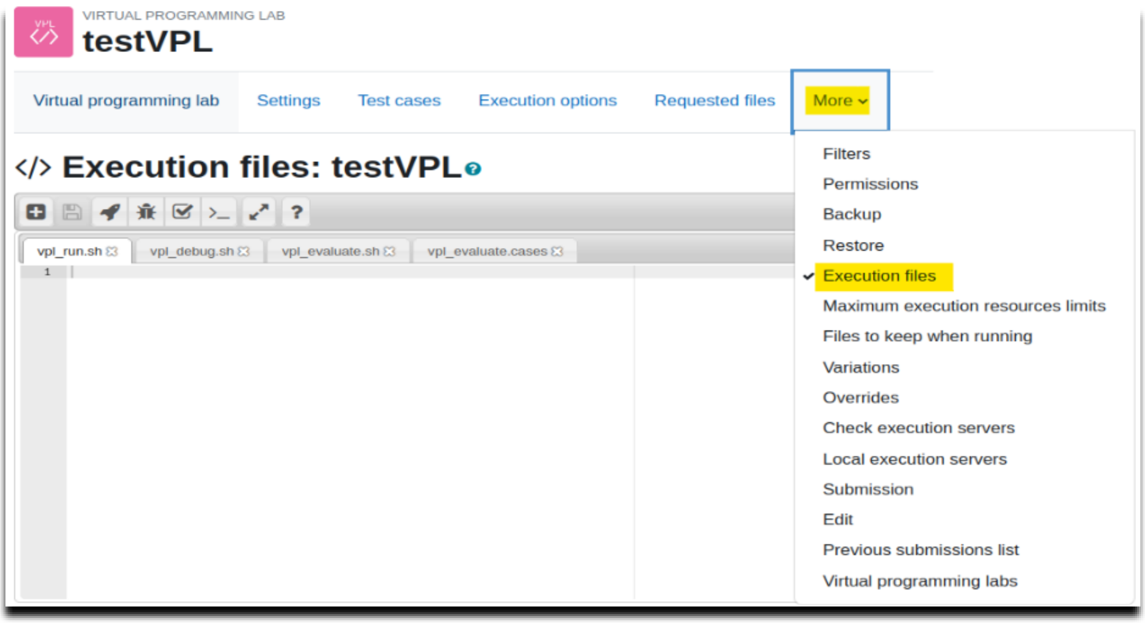
Evaluate just on submission: Yes

Automatic grade: Yes

save options

Step 8: Configure Execution Files

1. Within the assignment settings, click on the "More" tab.
2. Choose "Execution Files" from the options.
3. Insert all "Execution Files" needed like data files, scripts provided, scripts you will be using in your program , etc.


VIRTUAL PROGRAMMING LAB

testVPL

Virtual programming lab | Settings | Test cases | Execution options | Requested files | More

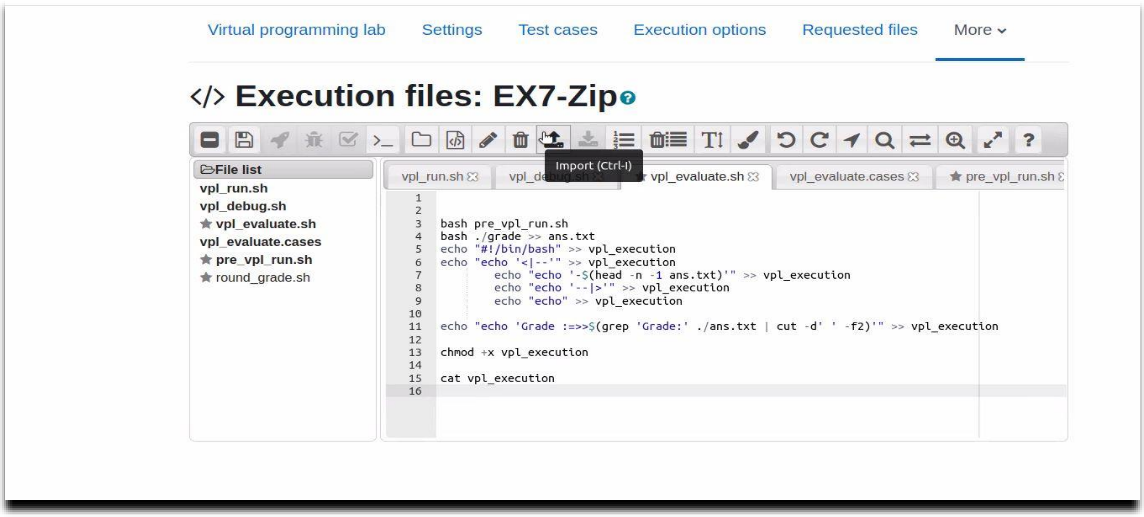
Execution files: testVPL

vpl_run.sh | vpl_debug.sh | vpl_evaluate.sh | vpl_evaluate.cases

Step 9: Choose Execution Files

For each programming language and submission type:

1. Select the appropriate zip file from provided directories.
2. Import chosen files into VPL Execution files for proper execution and evaluation. (Note: For Python programs with tests, ensure test file names end with _test.py).



```


bash pre_vpl_run.sh
bash ./grade >> ans.txt
echo "#!/bin/bash" >> vpl_execution
echo "echo '<|--'" >> vpl_execution
        echo "echo '-$(head -n -1 ans.txt)'" >> vpl_execution
        echo "echo '--|>'" >> vpl_execution
        echo "echo" >> vpl_execution

echo "echo 'Grade :=>>$(grep 'Grade:' ./ans.txt | cut -d' ' -f2)'" >> vpl_execution

chmod +x vpl_execution

cat vpl_execution

```

Step 10: Customize Script Files

1. Modify default directory file names within the pre_vpl_run.sh script imported earlier.
2. Ensure requested file name matches the chosen name from Step 6 (requested file).
3. For Debugging see Step 12.

Step 11: Teacher Review and Automatic Grading

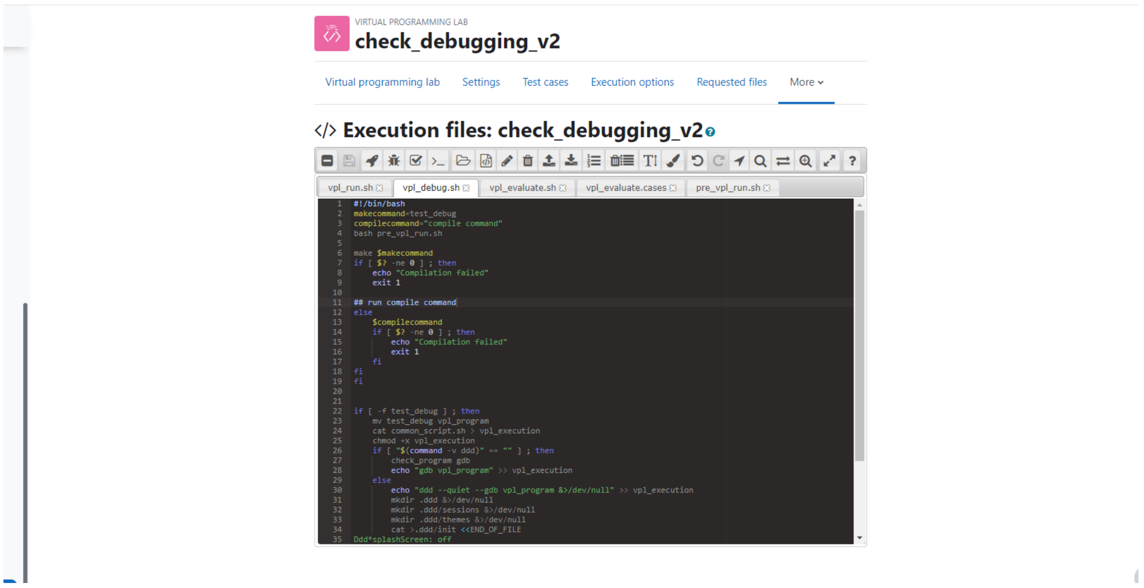
1. The system automatically grades student submissions based on self-evaluation.
2. Access student submissions by clicking "Setting Click" and then "Submission View."

Additional Resources: For a comprehensive demonstration, watch our instructional video: [VPL Assignment Upload Video]()

Step 12: Debugging

Debugging is a crucial aspect of programming, and the VPL system offers an efficient way to facilitate this process. To initiate debugging, follow these steps:

1. **Access the Debug Script**: For debugging purposes, you can utilize a script called **vpl_debug.sh**. This script prepares the program for debugging. It achieves this by triggering either the **make** command or by specifying the compile command using the GCC compiler as a string in the appropriate variable within the debug file.
2. **Compile Command Configuration**: Suppose, for instance, your make file includes an option for debugging labeled "**test_debug**". In this case, you would set the **"makecommand**" variable to "**test_debug**". Subsequently, the execution server will compile the program and initiate it in the GDB debugger directly within the user's browser. This integration utilizes the Data Display Debugger (DDD), which offers a graphical user interface for smooth and effective debugging.



```
#!/bin/bash
makecommand=test_debug
compilecommand="compile command"
bash pre_vpl_run.sh

make $makecommand
if [ $? -ne 0 ] ; then
    echo "Compilation failed"
    exit 1

## run compile command
else
    $compilecommand
    if [ $? -ne 0 ] ; then
        echo "Compilation failed"
        exit 1
    fi
fi
fi


if [ -f test_debug ] ; then
    mv test_debug vpl_program
    cat common_script.sh > vpl_execution
    chmod +x vpl_execution
    if [ "$(command -v ddd)" == "" ] ; then
        check_program gdb
        echo "gdb vpl_program" >> vpl_execution
    else
        echo "ddd --quiet --gdb vpl_program &>/dev/null" >> vpl_execution
        mkdir .ddd &>/dev/null
        mkdir .ddd/sessions &>/dev/null
        mkdir .ddd/themes &>/dev/null
        cat >.ddd/init <<END_OF_FILE
Ddd*splashScreen: off
```

3. **Step 3: Submit Your Solution and Debug Your Program:**

   Once you've made the necessary edits to the files required for the debugging process, the next step is to submit an example solution. This solution will be used to run a test to identify and fix any issues in your program.

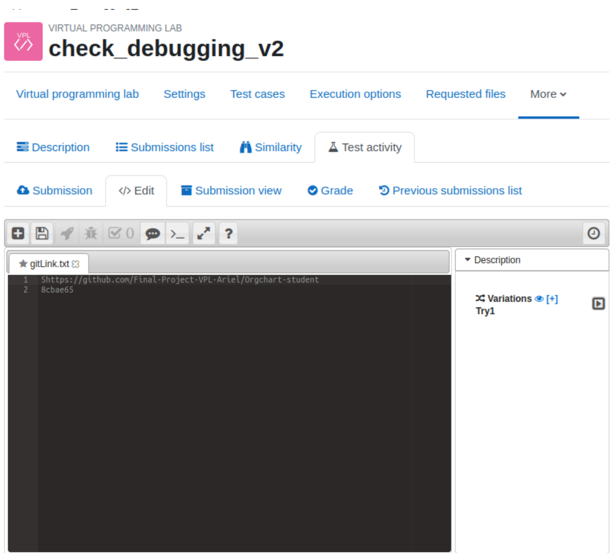
   Here's how you can do it:

   Access the Test Activity Tab: You'll find the test activity tab on the assignment's homepage.

   Two Options - Edit and Submission: In the upcoming window, you'll see two buttons: "Edit" and "Submission."

   Edit: Use the edit button if you want to modify the current file or create a new one using an editor. This is helpful if you need to make any last-minute changes to your solution.

   Submission: If your solution is ready, click on the submission button. You can upload your solution file in either zip format or as a text file containing a GitHub URL. This is the version of your solution that will be tested.



```
Shttps://github.com/Final-Project-VPL-Ariel/Orgchart-student
8cbae65
```

By following these steps, you can seamlessly integrate debugging capabilities into your programming assignments, enhancing the learning experience and facilitating the identification and resolution of programming errors.

Late Submission Option

In cases where students need to submit their assignments after the designated due date, while still accounting for a grade reduction, the override interface within the assignment settings can be employed. Here's how you can implement this option:

1. **Access the Assignment Page**: Navigate to the assignment's home page in your course's platform.
2. **Access the Override Interface**: Beneath the tabs on the menu, locate "More" and proceed to "Override."
3. **Create a New Override**: Initiate a new override by selecting the specific student for whom you wish to apply this modification. Then, establish a new due date. Additionally, you have the option to implement a grade reduction for the late submission. There are two ways to accomplish this:

a. **Direct Override Entry**: Within the override settings, directly enter the grade reduction value. This method is suitable for a fixed reduction.

b. **Fail Penalty File Editing**: Alternatively, you can modify the **fail_penalty.txt** file and upload it to the execution files.

The file can be found on this repository under scripts/bash-utils.

The format for this involves each line containing the number of days past the original due date, followed by a colon separator and the corresponding grade reduction points. This approach allows for gradual reduction based on days elapsed.

By utilizing the late submission option and integrating grade reduction measures, you accommodate various scenarios while maintaining a structured and equitable assessment process.